How-Tos
Wing IDE Personal

Wingware
www.wingware.com

Version 2.1.3
October 12, 2006

Dies ist eine Sammlung von HOW-TOs, die dafür bestimmt ist, das Starten von Wing IDE mit speziellen Werkzeugen und für fortgeschrittene Entwicklungsaufgaben zu erleichtern.

Inhalt

Wing IDE Schnellstart-Anleitung

- Installation von Python und Wing IDE
- Ein Projekt einrichten
- Hauptfunktionen
- Verwandte Dokumente

Verwendung von Wing IDE mit wxPython

- Installation und Konfiguration
- Testausführung des Debuggers
- Testausführung des Source-Browsers
- Verwendung eines GUI-Builders
- Verwandte Dokumente

Using Wing IDE with PyGTK

- Installation and Configuration
- Auto-completion and Source Assistant
- Using a GUI Builder
- Details and Notes
- Related Documents

Verwendung von Wing IDE mit PyQt

- Installation and Konfiguration
- Testausführung des Debuggers
- Testausführung des Source-Browsers
- Verwendung eines GUI-Builders
- Tipps, wie Sie den Debug-Prozess ansprechbar halten
- Verwandte Dokumente

Using Wing IDE with Zope

- Quick Start on a Single Host

- Starting the Debugger
- Test Drive Wing IDE
- Setting Up Auto-Refresh
- Setting up Remote Debugging
- Trouble Shooting Guide
- Related Documents

Verwendung von Wing IDE mit Plone

- Hinweise zur Leistungsfähigkeit
- Verwandte Dokumente

Verwendung von Wing IDE mit mod_python

- Schnellstart
- Beispiel
- Verwandte Dokumente

Web-CGIs mit Wing IDE debuggen

Wing IDE for OS X

- Usage Tips
- Known Problems
- Related Documents

Using Wing IDE with pygame

- Debugging pygame
- Related Documents

Große Werte und Strings im Debugger verarbeiten

C/C++ und Python zusammen debuggen

Erweiterungsmodule in Linux/Unix debuggen

Anwendungen debuggen, die XGrab* benutzen

Nicht-Python-Hauptschleifen debuggen

- Unterstützte Nicht-Python-Hauptschleifen
- Mit Nicht-Python-Hauptschleifen arbeiten
- Nicht-Python-Hauptschleifen Internals

Unterstützung für Nicht-Python-Hauptschleifen
Support für Nicht-Python-Hauptschleifen schreiben
Debuggen mit Python, das mit --with-pydebug kompiliert wurde
Debugging Code Running Under Py2exe

Wing IDE Schnellstart-Anleitung

Dies ist eine kurze Anleitung für diejenigen, die Wing IDE so schnell wie möglich zum Laufen bringen möchten. Eine umfassendere Einführung bietet unser **Tutorial**.

Außerdem verfügbar: Schnellstart-Anleitungen speziell für die **OS X Installation**, für **Zope**, **Plone**, **wxPython**, **PyQt** und **mod_python**.

> Wir begrüßen Feedback und Fehlerberichte. Beide können direkt in Wing IDE mit den Einträgen `Feedback einreichen` und `Fehlerbericht einreichen` im Hilfe-Menü eingereicht werden; sie können uns auch eine E-Mail an support at wingware.com senden.

Installation von Python und Wing IDE

Sowohl Python als auch Wing IDE müssen installiert sein. Die Wing IDE Executable heißt `wing-personal2.1`. Zusätzliche Informationen finden Sie unter **Installation**, **Ausführung des IDEs** und **Installation Ihrer Lizenz**.

Ein Projekt einrichten

Wing startet am Anfang mit einer leeren, unbenannten Projektdatei. Damit Sie Wing's Funktionen vollständig ausschöpfen können, müssen Sie eine Projektdatei folgendermaßen einrichten:

- Verwenden Sie `Verzeichnisbaum hinzufügen` und andere Einträge aus dem Menü `Projekt`, um Source-Dateien zu Ihrem Projekt hinzuzufügen.
- Nutzen Sie `Projekteigenschaften` aus dem Menü `Projekt`, um den `PYTHONPATH` zu bestimmen und um die spezielle Python-Version auszuwählen, die für Ihr Projekt verwendet werden soll.

Diese zwei Schritte informieren Wing darüber, wie es Importe in Ihrem Code handhaben soll, so dass es Ihre Source-Basis ausfindig machen und analysieren kann. Dies betreibt Wing's Source-Browser, Auto-Vervollständiger, Source-Assistent und die Code-Navigationsfunktionen.

- Speichern Sie Ihr Projekt auf der Festplatte.

Hinweis: In Abhängigkeit von der Größe der Code-Basis, die Sie zu Ihrem Projekt hinzufügen, kann Wing für mehrere Minuten beträchtliche CPU-Zeit verbrauchen, um Ihren Source-Code zu analysieren. Sobald dies erledigt ist, sollte Wing selbst auf langsameren Maschinen mit einer schnellen und gut ansprechenden Oberfläche laufen.

Siehe **Debug-Eigenschaften** und **Source-Code-Analyse** für Einzelheiten.

Hauptfunktionen

Sie sind jetzt bereit, das Code Schreiben und Debuggen zu beginnen. Die meisten Funktionen von Wing sind in der Benutzeroberfläche selbst erklärend. Die **Wing Tipps** liefern nützliche Hinweise zum Gebrauch des IDEs.

Diese Funktionen sollten Sie auf jeden Fall ausprobieren, während Sie Wing IDE beurteilen:

- *Anpassbare Benutzeroberfläche* -- Viele Optionen stehen in den **Einstellungen** zur Verfügung. Sie können Werkzeugfelder teilen und die Werkzeuge in diesen umherschieben. Ein rechter Mausklick auf die Notizbuchreiter liefert weitere Optionen oder Sie können das Menü `Fenster` verwenden, um Werkzeuge in einem separaten Fenster anzulegen. Ihre Konfiguration wird in der Projektdatei gespeichert.

- *Auto-Vervollständigung und Source-Assistent* -- Wing's Editor und das **Source-Assistent** Werkzeug stellen an den Kontext angepasste Vervollständigungsoptionen und Dokumentation bereit, wenn Sie Ihren Code bearbeiten.

- *Gehe zur Definition* -- Verfügbar in der Werkzeugleiste, im `Source`-Menü und durch einen rechten Mausklick auf Symbole im Editor.

- *Source-Index* -- Schneller Zugriff auf andere Teile einer Source-Datei über die Menüs, die sich oben im Source-Editor befinden.

- *Mini-Suche* -- Wing's leistungsfähige, tastaturgesteuerte Suchen- und Ersetzen-Einrichtung ist über das Menü `Bearbeiten` verfügbar. Die Verwendung der Tastaturbefehle, die im Menü angegeben sind, schlägt im unteren Bereich des Bildschirms den Eingabebereich für die Suche auf; geben Sie Ihre Suchzeichenkette ein und wiederholen die Tastaturkombinationen für eine wiederholte Suche vorwärts/rückwärts.

- *Suchmanager* -- Bietet das Suchen und Ersetzen für viele Dateien, sowie Wildcard- und reguläre Ausdruckssuche/-ersetzen. Verfügbar als **Suchen/Ersetzen** im Bereich der Werkzeuge.

- *Source-Browser* -- Modul- oder klassenorientierte Anzeige der Struktur Ihres Source-Codes. Sehen Sie sich sowohl den **Source-Browser** als auch den **Source-Assistenten** an, um umfassende Informationen über gewählte Symbole zu erhalten.

- *Grundlegendes Debuggen* -- Setzen Sie einen Haltepunkt und starten das Debuggen. Sie können die Werkzeuge **Stack-Daten** und **Module** verwenden, um Programmdaten zu prüfen und zu ändern. Wing unterscheidet zwischen schweren und nicht-schweren Exceptions zu der Zeit, zu der sie angetroffen werden, was Ihnen erlaubt, den aktuellen Programmstatus öfter zu überprüfen. Bedingte Haltepunkte sind ein leistungsfähiges Werkzeug für das Isolieren komplexer Fehler, da sie vor deren Auftreten stoppen. I/O des Debug-Prozesses wird in dem **integrierten I/O** Werkzeug (oder optional in einer externen Konsole) ausgegeben.

- *Debug-Test* -- Dieses Werkzeug stellt eine interaktive Python-Prompt bereit, die Code im aktuellen Stack-Frame des angehaltenen Debug-Prozesses ausführt. Der **Debug-Test** ist besonders hilfreich für das Isolieren und Verstehen komplexer Fehler sowie das Entwerfen von Code für die Behebung des Problems.

- *Werte beobachten* -- Geben Sie zu beobachtende Werte in das Werkzeug **Beobachten** ein oder klicken Sie mit der rechten Maustaste auf eine beliebige Datenansicht, um Werte über einen Zeitraum nach symbolischen Namen oder Objektverweis zu verfolgen.

- *Python-Shell* -- Diese **Python-Befehlsaufforderungszeile** lässt Sie Code in einem Sandbox-Prozess, der von Wing IDE und Ihrem Debug-Prozess isoliert ist, ausprobieren.

Verwandte Dokumente

Wing IDE stellt viele andere Optionen und Werkzeuge bereit. Weitere Informationen finden Sie hier:

- **Wing IDE Tutorial**, eine detaillierte, geführte Tour durch Wing IDE.
- **Wing IDE Benutzerhandbuch**, das Wing IDE eingehend beschreibt.
- **OS X Schnellstart-Anleitung**
- **Zope Schnellstart-Anleitung**
- **Plone Schnellstart-Anleitung**
- **wxPython Schnellstart-Anleitung**
- **PyQt Schnellstart-Anleitung**
- **mod_python Schnellstart-Anleitung**

Verwendung von Wing IDE mit wxPython

Wing IDE ist eine integrierte Entwicklungsumgebung für die Programmiersprache Python. Wing kann dazu verwendet werden, den Prozess des Schreibens und Debuggens von Code, der für das leistungsfähige wxPython, einem Toolkit für die Kreuzplattform GUI-Entwicklung, geschrieben ist, zu beschleunigen.

wxPython ist für den GUI-Entwickler eine gute Wahl. Es ist zur Zeit für MS Windows, Linux, Unix und Mac OS X verfügbar und stellt auf jeder dieser Plattformen das native Look-and-Feel bereit.

Obwohl Wing IDE zur Zeit keinen GUI-Builder für wxPython bereitstellt, bietet es die am weitesten fortgeschrittenen Debugger- und Source-Browser-Fähigkeiten, die für die Programmiersprache Python zur Verfügung stehen, und es kann mit anderen verfügbaren GUI-Buildern verwendet werden, was weiter unten beschrieben wird.

Installation und Konfiguration

Führen Sie die folgenden Schritte aus, um Wing IDE für die Verwendung mit wxPython einzustellen und zu konfigurieren.

- Installieren Sie Python und Wing. Sie benötigen eine bestimmte Python-Version, abhängig von der wxPython-Version, die Sie verwenden möchten. Wenn Sie Zweifel haben, lesen Sie das wxPython Getting Started Wiki. Lesen Sie auch die allgemeine **Wing IDE Schnellstart-Anleitung** für Installationsanweisungen.

- Installieren Sie wxPython. Lesen Sie die wxPython-Website Getting Started Wiki für Installationsanweisungen. Beachten Sie, dass Sie die Version von wxPython installieren müssen, die mit Ihrer Python-Version zusammenpasst, wie es auf der Download-Seite beschrieben ist.

- Starten Sie Wing IDE vom Startmenü in Windows oder geben Sie in Linux, OS X oder anderen Posix-Systemen „wing“ in die Befehlszeile ein.
- Wählen Sie aus dem Menü Source den Punkt `Analysestatistiken anzeigen`. Wenn die dort angezeigte Python-Version nicht mit der übereinstimmt, die Sie mit wxPython verwenden, dann gehen Sie zu Projekteigenschaften im Projektmenü und verwenden das Feld Python-Executable, um die richtige Python-Version auszuwählen.
- Öffnen Sie `Lib/site-packages/wx/demo/demo.py` in Wing IDE (innerhalb Ihrer Python-Installation platziert) und wählen Sie aus dem Projektmenü den Punkt `Aktuelle Datei hinzufügen` aus.
- Setzen Sie `demo.py` als Debug-Startpunkt für das Debuggen, indem Sie aus dem Menü Debuggen den Punkt `Aktuelle als Haupt-Debug-Datei einstellen` auswählen.
- Speichern Sie Ihr Projekt auf dem Laufwerk. Verwenden Sie einen Namen, der auf `.wpr` endet.

Testausführung des Debuggers

Sie sind jetzt bereit, den Debugger auszutesten. Um dies zu tun:

- Starten Sie das Debuggen mit dem Eintrag `Debuggen / Fortsetzen` aus dem Menü Debuggen. Entfernen Sie die Markierung des Kontrollkästchens `Diesen Dialog vor jedem Durchlauf anzeigen`, das am Ende des Dialogs erscheint, und wählen Sie dann `OK`.
- Die Demo-Anwendung wird starten. Wenn das Hauptfenster nicht nach vorn kommt, bringen Sie es von Ihrer Taskleiste oder Ihrem Fenstermanager nach vorn. Wenn Sie die verschiedenen Demos von dem Baum links von der wxPython-Demo-Anwendung ausprobieren, werden Sie manchmal sehen, dass Wing IDE im `Exceptions` Werkzeug des Debuggers Exceptions berichtet. Diese sind keine Programmfehler oder Störungen des IDE's, sondern werden von Wing's proaktivem Exception-Erkennungsalgorithmus verursacht, der nicht in den C und C++ Code, der diese Exceptions außerhalb des Debuggers behandelt, sehen kann.

  Um diese zu übergehen, wählen Sie `Diese Exception-Position ignorieren` im `Exceptions` Werkzeug und setzen die Ausführung fort. Sie werden 3-4 von diesen sehen, wenn Wing das erstem Mal auf sie trifft. Danach wird Ihre Ignorierliste im Projekt gespeichert, so dass Sie sie nie wieder sehen, selbst in zukünftigen Debug-Sitzungen. Im Nachhinein werden Sie von Wing's Fähigkeit, sofort am

Punkt der Exception anstatt nach dem Fakt anzuhalten, profitieren. Dies macht das Verstehen der Bedingungen, die zu einem Fehler führen, viel einfacher und beschleunigt die Debug-Bearbeitungszeit.

In wxPython 2.3.4.2 für Python 2.2 können Beispiele dieser Exceptions angesehen werden, indem Sie More Dialogs / ImageBrowser, More Dialogs / wxMultipleChoiceDialog und New since last release / Throbber aufschlagen.

- Öffnen Sie als nächstes `Lib/site-packages/wx/demo/ImageBrowser.py` in Wing IDE. Setzen Sie einen Haltepunkt an der ersten Zeile von `runTest()`, indem Sie auf den dunkelgrauen linken Seitenrand klicken. Gehen Sie in die laufende Demo-Anwendung und wählen More Dialogs / ImageBrowser aus. Wing wird an Ihrem Haltepunkt stoppen.

- Wählen Sie `Stack-Daten` aus dem Menü Werkzeuge. Schauen Sie im Stack im Popup obem im Fenster und den Lokalen und Globalen, die darunter angezeigt werden, nach dem gewählten Stack-Frame. Es kann sein, dass Sie beim Anzeigen von Werten etwas Trägheit bemerken (einige Sekunden). Dies liegt an dem weitverbreiteten Gebrauch von `from wx import *` in wxPython-Code, der eine große Anzahl von Symbolen in den globalen Namensbereich importiert. Es hängt von der Geschwindigkeit Ihrer Maschine ab.

- Wahlen Sie `Debug-Test` aus dem Menü Werkzeuge. Das ist eine interaktive Befehlsaufforderung, die Sie Ausdrücke tippen lässt oder sogar Werte im Kontext des Stack-Frames, der im Debugger-Fenster ausgewählt ist, ändern lässt, wenn Ihr Programm an einer Exception anhält oder stoppt. Es ist ein sehr leistungsfähiges Debug-Werkzeug.

- Werfen Sie auch einen Blick auf diese Werkzeuge, die im Menü Werkzeuge bereitstehen:

  - `I/O` -- zeigt die Ausgabe des Debug-Prozesses an und verarbeitet die Tastatureingaben an den Debug-Prozess, wenn vorhanden.
  - `Exceptions` -- zeigt Exceptions an, die im Debug-Prozess auftreten.
  - `Module` -- durchsucht Daten für alle Module in `sys.modules`.
  - `Beobachten` -- beobachtet Werte, die von anderen Wertansichten ausgwählt wurden (durch Rechtsklicken und Auswählen eines der `Beobachten` Einträge) und erlaubt, die Eingabe von Ausdrücken, um sie im akutellen Stack-Frame zu bewerten.

Testausführung des Source-Browsers

Vergessen Sie nicht, einen Blick auf Wing's leistungsfähigen Source-Browser zu werfen:

- Fügen Sie das Paket `Lib/site-packages/wx` (innerhalb Ihrer Python-Installation) zu Ihrer Projektdatei hinzu, und zwar mit dem Punkt `Paket hinzufügen` im Menü Projekt.

- Nachdem Sie das ausgeführt haben, wird Wing 20 Sekunden oder länger 100% Ihrer CPU verbrachen, abhängig von der Geschwindigkeit Ihrer Maschine. Während dies durchgeführt wird, können Sie bereits den Source-Browser vom Menü Werkzeuge aufschlagen. Seien Sie einfach geduldig, wenn Dinge am Anfang ein wenig träge erscheinen; es gibt jede Menge Python-Code, den Wing analysieren muss. Wenn die anfängliche Analyse einmal erledigt ist, wird Wing wieder reagieren, da die Ergebnisse gespeichert sind (ein ähnlicher aber kürzerer Effekt ist zu beobachten, wenn Wing neu gestartet wird, da es den Analyse-Laufwerkspeicher liest).

- Wählen Sie den Modus `Alle Klassen` vom oberen Teil des Source-Browsers. Das ist im allgemeinen die beste Ansicht für wxPython. Auf langsameren Maschinen kann die Nach-Modul-Ansicht auch träge erscheinen, was aus der großen Anzahl von Symbolen resultiert, die aufgrund der Verwendung von `from wx import *` in wxPython-Source auf der Modulebene gefunden werden. Wenn Sie die Nach-Modul-Ansicht nutzen, ist es hilfreich, die Selektion des Kontrollkästchens `Geerbt` zu entfernen.

- Verwenden Sie das Menü, das mit einem rechten Mausklick aufgeschlagen wird, um zu Basisklassen zu zoomen. Im Allgemeinen schlägt ein rechter Mausklick spezielle Menüs zu dem Werkzeug, auf das geklickt wurde, auf.

- Verwandt zum Source-Browser ist die Fähigkeit der Auto-Vervollständigung in Wing's Source-Editor. Versuchen Sie, in einer der wxPython-Source-Dateien etwas einzugeben, und Sie werden sehen, dass der Auto-Vervollständiger erscheint. Die Tab-Taste vervollständigt den zur Zeit ausgewählten Eintrag. Sie können allerdings auch mit der Eingabetaste vervollständigen, wenn Sie die Einstellung **Mit der Eingabetaste automatisch vervollständigen** setzen. Lesen Sie die **Wing IDE Schnellstart-Anleitung** für Informationen zu anderen allgemein verwendeten Einstellungen.

  Hinweis: In Abhängigkeit von der Geschwindigkeit Ihrer Maschine kann der Auto-Vervollständiger zunächst träge erscheinen, was wiederum auf die große Anzahl von Symbolen, die in die meisten wxPython-Dateien mit `from wx import *` importiert werden, zurückzuführen ist. Dies wird jedoch nur einmal pro Wing IDE Sitzung auftreten.

- Werfen Sie auch einen Blick auf das Werkzeug `Source-Assistent` im Menü Werkzeuge. Es stellt zusätzliche Informationen über Source-Konstrukte im aktiven Source-Editor bereit, wenn der Einfügungscursor oder die -auswahl verschoben wird.

Verwendung eines GUI-Builders

Wing IDE enthält zur Zeit keinen GUI-Builder für wxPython, aber es kann mit anderen Werkzeugen verwendet werden, wie zum Beispiel Boa Constructor, das einen GUI-Builder bereitstellt, aber nicht die kraftvolle Leistung von Wing's Debugger und Source-Browser besitzt.

Um einen externen GUI-Builder zu verwenden, **konfigurieren Sie Wing auf das automatische Neuladen von Dateien**, die vom GUI-Builder verändert werden.

Dann können Sie Wing IDE und Ihren GUI-Builder zur gleichen Zeit ausführen und mit beiden in einer fast nahtlosen Weise arbeiten.

Ein Hinweis: Da Python sich selbst so gut für datengesteuerten Code zur Verfügung stellt, werden Sie es noch einmal überdenken wollen, einen GUI-Builder für einige Aufgaben zu verwenden. In vielen Fällen machen es die Selbstbeobachtungsfunktionen von Python möglich, allgemeinen GUI-Code zu schreiben, den Sie zur schnellen Erstellung von Benutzeroberflächen, basierend auf Modellen Ihrer Daten und Ihrer Anwendung, verwenden können. Dies kann sehr viel effizienter sein, als die Verwendung eines GUI-Builders zur Erstellung von individellen Menüs und Dialogen per Hand. Im Allgemeinen neigen per Hand codierte GUIs dazu, wartungsfreundlicher zu sein.

Verwandte Dokumente

Wing IDE stellt viele andere Optionen und Tools bereit. Weitere Informationen finden Sie hier:

- **Wing IDE Benutzerhandbuch**, das Wing IDE detailliert beschreibt.
- wxPython Getting Started Seite, welche viele zusätzliche Informationen für wxPython-Programmierer enthält.
- **Wing IDE Schnellstart-Anleitung**, die zusätzliche grundlegende Informationen darüber enthält, wie Sie Wing IDE zum Laufen bringen.

Using Wing IDE with PyGTK

Wing IDE is an integrated development environment for the Python programming language. Wing can be used to speed up the process of writing and debugging code that is written for PyGTK and GTK+, a mature open source GUI development toolkit.

PyGTK is currently available for Linux/Unix, MS Windows, and Mac OS X (requires X11 Server). Like **PyQt** and unlike **wxPython**, PyGTK runs on the same (GTK-provided) widget implementations on all platforms. Themes can be used to approximate the look and behavior of widgets on the native OS. It is also possible to display native dialogs like the Windows file and print dialogs along side GTK windows. While PyGTK does not offer perfect native look and feel, its provides excellent write-once-works-anywhere capability even in very complex GUIs. Wing IDE is itself written using PyGTK.

Other advantages of PyGTK include: (1) high quality anti-aliased text rendering, (2) powerful signal-based architecture that, among other things, allows subclassing C classes in Python, (3) multi-font text widget with embeddable sub-widgets, (4) model-view architecture for list and tree widgets, and (5) a rich collection of widgets and stock icons.

While Wing IDE does not currently provide a GUI builder for PyGTK, it does provide the most advanced debugger and source browser capabilities available for the Python programming language and it can be used with other available GUI builders, as described below.

Installation and Configuration

Take the following steps to set up and configure Wing IDE for use with PyGTK:

- Install Python and Wing. See the generic **Wing IDE Quickstart Guide** for installation instructions.

- Install GTK and PyGTK. If you are on Linux, you may already have one or both

installed, or you may be able to install them using your distribution's package manager. Otherwise, check out the gtk website and pygtk website.

- Start Wing from the Start menu on Windows or by typing „wing“ on the command line on Linux, OS X, or other Posix systems.
- Select Show Analysis Stats from the Source menu and if the Python version reported there doesn't match the one you're using with PyGTK, then select Project Properties from the Project menu and use the Python Executable field to select the correct Python version.
- Add some files to your project, and set the main entry point using the Set Main Debug File item in the Debug menu. Save the project file to disk.
- You should now be able to debug your PyGTK application from within Wing. If you see ImportErrors on the PyGTK modules, you will need to add Python Path information in the Debug tab of Project Properties, accessed from the Project menu.

Auto-completion and Source Assistant

To obtain auto-completion options and call signature information in Wing's Source Assistant, you will need to run a script that converts from PyGTK's defs files into Python interface files that Wing's source analyser can read.

- Download the pygtk_to_pi.py script and the PyGTK sources for your version of PyGTK if you don't already have them.
- Run as described within the script to produce a `*.pi` file for each `*.so` or `*.pyd` file in the PyGTK sources.
- Copy these `*.pi` files into the installed copy of PyGTK, so they sit next to the compiled `*.so` or `*.pyd` extension module file that they describe.
- Wing should now provide auto-completion and Source Assistant information when you `import gtk` and type `gtk.` in the editor.

Using a GUI Builder

Wing IDE doesn't currently include a GUI builder for PyGTK but it can be used with other tools, such as glade.

To use an external GUI builder, **configure Wing to automatically reload files** that are altered by the GUI builder.

Then you can run Wing IDE and your GUI builder at the same time, working with both in an almost seamless manner.

A Caveat: Because Python lends itself so well to writing data-driven code, you may want to reconsider using a GUI builder for some tasks. In many cases, Python's introspection features make it possible to write generic GUI code that you can use to build user interfaces on the fly based on models of your data and your application. This can be much more efficient than using a GUI builder to craft individual menus and dialogs by hand. In general hand-coded GUIs also tend to be more maintainable.

Details and Notes

- Building GTK from sources can be a challenge. Wingware has developed some build support scripts which we can provide on request to support at wingware dot com (these are slated for release as open source soon). We also have patches that allow GTK to be relocated after building on Linux/Unix.
- Native look and feel on Windows is provided by the gtk-wimp theme. If you plan to deploy on Windows, you may wish to contact us to obtain our latest performance patches for GTK on Windows.

Related Documents

Wing IDE provides many other options and tools. For more information:

- **Wing IDE Reference Manual**, which describes Wing IDE in detail.
- **Wing IDE Quickstart Guide** which contains additional basic information about getting started with Wing IDE.

Verwendung von Wing IDE mit PyQt

Wing IDE ist eine integrierte Entwicklungsumgebung für die Programmiersprache Python. Wing kann dazu verwendet werden, den Prozess des Schreibens und Debuggens von Code, der für PyQt, einem Toolkit für die Kreuzplattform GUI-Entwicklung, geschrieben ist, zu beschleunigen.

PyQt ist eine kommerzielle GUI-Entwicklungsumgebung, die auf Windows, Linux/Unix, Mac OS and dem Sharp Zauraus mit einem nativen Look-and-Feel läuft. Es ist auch kostenlos für nicht-kommerzielle Nutzer in Windows und für Open Source Entwickler in Linux/Unix erhältlich. Die Lizenzierung erfolgt pro Entwickler, ohne Kosten für jedes eingerichtete Produkt (sogar für kommerzielle Produkte).

Obwohl Wing IDE zur Zeit keinen GUI-Builder für PyQt bereitstellt, bietet es die am weitesten fortgeschrittenen Debugger- und Source-Browser-Fähigkeiten, die für die Programmiersprache Python zur Verfügung stehen, und es kann mit anderen verfügbaren GUI-Buildern verwendet werden, was weiter unten beschrieben wird.

Installation and Konfiguration

Führen Sie die folgenden Schritte aus, um Wing IDE für die Verwendung mit PyQt einzustellen und zu konfigurieren:

- Installieren Sie Python und Wing. Versichern Sie sich auf der PyQt Download-Seite, dass Sie eine Python-Version installieren, die mit der von Ihnen verwendeten Version von PyQt funktioniert. PyQt 3.5 funktioniert zum Beispiel mit allen Python-Versionen zwischen 1.5.2 und 2.2.x. Die allgemeine **Wing IDE Schnellstart-Anleitung** stellt Installationsanweisungen für Wing bereit.

- Installieren Sie Qt von Trolltech. Sie müssen entweder eine Entwickler-Lizenz erwerben oder ein nicht-kommerzielles Paket für Windows oder Linux/Unix herun-

terladen. Der einfachste Weg, den Download-Bereich zu finden, ist nach „download qt“ auf deren Website zu suchen.

- Installieren Sie PyQt von dem Riverbank PyQt Download-Bereich.

- Starten Sie Wing IDE vom Startmenü in Windows oder geben Sie in Linux, OS X oder anderen Posix-Systemen „wing“ in die Befehlszeile ein.
- Wählen Sie aus dem Menü Source den Punkt `Analysestatistiken anzeigen`. Wenn die dort angezeigte Python-Version nicht mit der übereinstimmt, die Sie mit PyQt verwenden, dann gehen Sie zu den Projekteigenschaften im Projektmenü und verwenden das Feld Python-Executable, um die richtige Python-Version auszuwählen.
- Öffnen Sie `PyQt/Examples/widgets.py` in Wing IDE (innerhalb Ihrer Python-Installation platziert) und wählen aus dem Projektmenü den Punkt `Aktuelle Datei hinzufügen` aus.
- Setzen Sie `widgets.py` als Debug-Startpunkt für das Debuggen, indem Sie aus dem Projektmenü den Punkt `Aktuelle als Haupt-Debug-Datei einstellen` auswählen.
- Speichern Sie Ihr Projekt auf dem Laufwerk. Verwenden Sie einen Namen, der auf `.wpr` endet.

Testausführung des Debuggers

Sie sind jetzt bereit, den Debugger auszutesten. Um dies zu tun:

- Starten Sie das Debuggen mit dem Eintrag `Debuggen / Fortsetzen` aus dem Menü Debuggen. Entfernen Sie die Selektion des Kontrollkästchens `Diesen Dialog vor jedem Durchlauf anzeigen`, das am Ende des Dialogs erscheint, und wählen Sie dann `OK`.
- Die Demo-Anwendung wird starten. Wenn das Hauptfenster nicht nach vorn kommt, bringen Sie es von Ihrer Taskleiste oder Ihrem Fenstermanager nach vorn.
- Öffnen Sie als nächstes `PyQt/Examples/widgets.py` (innerhalb Ihrer Python-Installation) in Wing IDE. Setzen Sie einen Haltepunkt zum Ende hin der AnalogClock's `paintEvent()` Methode. Während der nächsten Uhrenaktualisierung, die einmal pro Minute passiert, wird Wing IDE an diesem Punkt anhalten. Sie können auch veranlassen, dass der Haltepunkt eher erreicht wird, indem Sie die Uhr mit einem anderen Fenster verdecken und dann wieder aufdecken.

- Verwenden Sie das Werkzeug Stack-Daten, um im Stack und den Lokalen und Globalen nach dem ausgewählten Stack-Frame zu schauen.
- Wählen Sie `Debug-Test` aus dem Menü Werkzeuge. Dies ist eine interaktive Befehlsaufforderung, die Sie Ausdrücke tippen lässt oder sogar Werte im Kontext des Stack-Frames, der im Debugger-Fenster ausgewählt ist, ändern lässt, wenn Ihr Programm an einer Exception anhält oder stoppt. Es ist ein sehr leistungsfähiges Debug-Werkzeug.
- Werfen Sie auch einen Blick auf diese Werkzeuge, die im Menü Werkzeuge zur Verfügung stehen:
  - `I/O` -- zeigt die Ausgabe des Debug-Prozesses an und verarbeitet Tastatureingaben an den Debug-Prozess, wenn vorhanden.
  - `Exceptions` -- zeigt Exceptions an, die im Debug-Prozess auftreten.
  - `Module` -- durchsucht Daten für alle Module in `sys.modules`.
  - `Beobachten` -- beobachtet Werte, die von anderen Wertansichten ausgewählt wurden (durch Rechtsklicken und Auswählen eines der `Beobachten` Einträge) und erlaubt das Eingeben von Ausdrücken, um sie im aktuellen Stack-Frame zu bewerten.

Wenn Sie die verschiedenen Demos für PyQt ausprobieren, werden Sie manchmal sehen, dass Wing IDE anhält und Exceptions in dem `Exceptions` Werkzeug des Debuggers berichtet. Es gibt ein paar Fehler in einigen Versionen von PyQt's Demos, so dass Wing diese erfasst, wenn sie auftreten.

Es ist außerdem hilfreich zu wissen, dass Wing IDE manchmal (aber sehr selten in PyQt-Anwendungen) Exceptions berichtet, die Sie außerhalb des Debuggers nicht sehen. Diese sind keine Programmfehler, sondern werden durch Wing's proaktiven Exception-Erkennungsalgorithmus verursacht, der nicht in den C und C++ Code, der diese Exceptions außerhalb des Debuggers behandelt, sehen kann.

Um diese Arten von Exceptions zu übergehen, wählen Sie `Diese Exception-Position ignorieren` im `Exceptions` Werkzeug des Debuggers aus und setzen die Ausführung fort. Ihre Ignorierliste wird im Projekt gespeichert, so dass Sie sie nie wieder sehen werden, selbst in zukünftigen Debug-Sitzungen. Im Nachhinein werden Sie von Wing's Fähigkeit, sofort am Punkt der Exception anstatt nach dem Fakt anzuhalten, profitieren. Dies macht das Verstehen der Bedingungen, die zu einem Fehler führen, viel einfacher und beschleunigt die Debug-Bearbeitungszeit.

Testausführung des Source-Browsers

Vergessen Sie nicht, einen Blick auf Wing's leistungsfähigen Source-Browser zu werfen:

- Fügen Sie das Paket `Lib/site-packages` (innerhalb Ihrer Python-Installation) zu Ihrem Projekt hinzu, und zwar mit dem Punkt `Paket hinzufügen` im Menü Projekt. Fügen Sie außerdem den Verzeichnisbaum PyQt (auch innerhalb Ihrer Python-Installation) zu Ihrer Projektdatei hinzu; verwenden Sie dazu den Eintrag `Verzeichnisbaum hinzufügen` aus dem Menü Projekt. Sie sollten jetzt vier Einträge in der höchsten Ebene des Projektfensters sehen.

- Schlagen Sie als nächstes den `Source-Browser` aus dem Menü Werkzeuge auf. Sie können den Ansichtstil oben im Fenster auswählen: `Nach Modulen`, um das Laufwerklayout zu durchsuchen, `Klassenhierarchie`, um nur Basisklassen auf der höchsten Ebene zu sehen und `Alle Klassen`, um eine Liste von allen Klassen nach Namen zu sehen. Das Menü `Optionen` auf der rechten Seite filtert, welche Symboltypen im Browser angezeigt werden.

- Das Klicken auf den Browser wird den entsprechenden Source-Code in dem Source-Editor-Bereich anzeigen. Beachten Sie, dass Dateien automatisch geschlossen werden, wenn Sie woanders durchsuchen, es sei denn, sie waren bereits geöffnet, Bearbeitungen wurden vorgenommen oder Sie klicken auf den Stick-Pin in der oberen rechten Ecke des Editorbereiches, um zu bestimmen, dass der Editor geöffnet bleiben soll, bis er explizit geschlossen wird.

- Verwenden Sie das Menü, das mit einem rechten Mausklick aufgeschlagen wird, um zu Basisklassen zu zoomen. Im Allgemeinen schlägt ein rechter Mausklick spezielle Menüs zu dem Werkzeug, auf das geklickt wurde, auf.

- Verwandt zum Source-Browser ist die Fähigkeit der Auto-Vervollständigung in Wing's Source-Editor. Versuchen Sie, in einer der PyQt-Source-Dateien etwas einzugeben, und Sie werden sehen, dass der Auto-Vervollständiger erscheint. Die Tab-Taste vervollständigt den zur Zeit ausgewählten Eintrag. Sie können allerdings auch mit der Eingabetaste vervollständigen, wenn Sie die Einstellung **Mit der Eingabetaste automatisch vervollständigen** setzen. Lesen Sie die **Wing IDE Schnellstart-Anleitung** für Informationen zu diesen und anderen allgemein verwendeten Einstellungen.

- Werfen Sie auch einen Blick auf das Werkzeug `Source-Assistent` im Menü Werkzeuge. Es stellt zusätzliche Informationen über Source-Konstrukte im aktiven Source-Editor bereit, wenn der Einfügungscursor oder die -auswahl verschoben wird.

Verwendung eines GUI-Builders

Wing IDE enthält zur Zeit keinen GUI-Builder für PyQt, aber es kann mit anderen Tools verwendet werden, wie zum Beispiel Black Adder, das einen GUI-Builder bereitstellt,

aber nicht die kraftvolle Leistung von Wing IDE's Debugger und Source-Browser besitzt. Ein anderer GUI-Builder für PyQt ist Qt Designer, der sprachenunabhängige UI-Dateien ausgibt, die unter Verwendung von PyQt's `pyuic` Hilfsprogramm in Python umgewandelt werden können.

Um einen externen GUI-Builder zu verwenden, **konfigurieren Sie Wing auf das automatische Neuladen von Dateien**, die vom GUI-Builder verändert werden.

Dann können Sie Wing IDE und Ihren GUI-Builder zur gleichen Zeit ausführen und mit beiden in einer fast nahtlosen Weise arbeiten.

Ein Hinweis: Da Python sich selbst so gut für datengesteuerten Code zur Verfügung stellt, werden Sie es noch einmal überdenken wollen, einen GUI-Builder für einige Aufgaben zu verwenden. In vielen Fällen machen es die Selbstbeobachtungsfunktionen von Python möglich, allgemeinen GUI-Code zu schreiben, den Sie zur schnellen Erstellung von Benutzeroberflächen, basierend auf Modellen Ihrer Daten und Ihrer Anwendung, verwenden können. Dies kann sehr viel effizienter sein, als die Verwendung eines GUI-Builders zur Erstellung von individellen Menüs und Dialogen per Hand. Im Allgemeinen neigen per Hand codierte GUIs dazu, wartungsfreundlicher zu sein; und das Qt-Widget-Set wurde speziell dafür entworfen, Hand-Codierung einfach zu machen.

Tipps, wie Sie den Debug-Prozess ansprechbar halten

Aufgrund von Fehlern in einigen PyQt-Versionen gibt es keinen Code innerhalb des Debuggers, der sicherstellt, dass PyQt-Debug-Prozesse ansprechbar zum Debugger bleiben, während das Debug-Programm läuft. Das bedeutet, dass Sie vielleicht nicht immer in der Lage sind, PyQt-Debug-Prozesse anzuhalten und dass der Debugger abschalten kann, wenn Sie versuchen, Haltepunkte hinzuzufügen oder bestimmte andere Debugger-Operationen ausführen, während die GUI-Anwendung läuft und kein Python-Code erreicht wird.

Diese Problem tritt nur auf, wenn überhaupt kein Python-Code erreicht wird. Daher ist es einfach, dies mit dem Folgenden zu umgehen, und zwar nachdem Ihre `QApplication` erstellt wurde und bevor Sie `exec_loop()` aufrufen:

```
# Hack to burn some Python bytecode periodically so Wing's
# debugger can remain responsive while free-running
import os
if os.environ.has_key('WINGDB_ACTIVE'):
  timer = QTimer()
  def donothing(*args):
    for i in range(0, 100):
      x = i
```

```
timer.connect(timer, SIGNAL("timeout()", donothing)
timer.start(500, 0)
```

Verwandte Dokumente

Wing IDE stellt viele andere Optionen und Tools bereit. Weitere Informationen finden Sie hier:

- **Wing IDE Benutzerhandbuch**, das Wing IDE detailliert beschreibt.
- PyQt Homepage, die Links zur Dokumentation bereitstellt.
- **Wing IDE Schnellstart-Anleitung**, die zusätzliche grundlegende Informationen darüber enthält, wie Sie Wing IDE zum Laufen bringen.

Using Wing IDE with Zope

Wing IDE can be used to develop and debug Python code running under Zope2, including Products, External Methods, file system-based Scripts and Zope itself. It is also useful for Zope-based frameworks like Plone (see **Plone Quickstart**).

Wing provides auto-completion, call tips, and other features that help to write, navigate, and understand Zope code. Wing's debugger can work with Zope's code reloading features to achieve a very short edit/debug cycle.

Limitations: Wing IDE cannot debug DTML or Python code that is not stored on the file system.

Security Warning: We advise against using the WingDBG product on production web servers. Any user connected to the Wing IDE debugger will (unavoidably) have extensive access to files and data on the system.

> **Note:** This guide is for Zope2 users. If you are using Zope3, please refer to **Debugging Externally Launched Code** in the users manual.

Quick Start on a Single Host

To use Wing IDE with Zope running on the same host as the IDE:

- **Install Zope** -- You can obtain Zope from zope.org. Version 2.5.1 or newer will work with Wing.
- **Install Wing IDE** -- You will need Wing IDE 2.1 or later. See **Installing** for details.
- **Configure Wing IDE** -- Start Wing, create or open the project you wish to use (from the `Project` menu). Then use the `Extensions` tab in `Project Properties` to enable **Zope2/Plone support** and to specify the **Zope2 Instance Home**

to use with the project. Wing will find your Zope installation by reading the file `etc/zope.conf` in the provided Zope instance.

Once you press `Apply` or `OK` in the Project Properties dialog, Wing will ask to install the WingDBG product and will offer to add files from your Zope installation to the project.

- **Configure the WingDBG Product** -- Start or restart Zope and log into http://localhost:8080/manage (assuming default Zope configuration). The Wing Debugging Service will be created automatically on startup; you can find it under the Control Panel of your server.

Starting the Debugger

Proceed to the Wing Debugger Service by navigating to the Control Panel, then selecting the 'Wing Debugging Service'. Click in the „Start“ button. The Wing IDE status area should display „Debugger: Debug process running“.

Note that you can configure WingDBG to start and connect to the IDE automatically when Zope is started from the Advanced configuration tab.

Problems? See the Trouble-Shooting Guide below.

Test Drive Wing IDE

Once you've started the debugger successfully, here are some things to try:

Run to a Breakpoint -- Open up your Zope code in Wing IDE and set a breakpoint on a line that will be reached as the result of a browser page load. Then load that page in your web browser using the port number displayed by the Zope Management Interface after you started the debugger. By default, this is 50080, so your URL would look something like this:

```
http://localhost:50080/Rest/Of/Usual/Url
```

Explore the Debugger Tools -- Take a look at these tools available from the Tools menu:

- `Stack Data` -- displays the stack, allows selecting current stack frame, and shows the locals and globals for that frame.

- `Debug Probe` -- lets you interact with your paused debug process using a Python shell prompt
- `Watch` -- watches values selected from other value views (by right-clicking and selecting one of the `Watch` items) and allows entering expressions to evaluate in the current stack frame
- `Modules` -- browses data for all modules in `sys.modules`
- `Exceptions` -- displays exceptions that occur in the debug process
- `Debug I/O` -- displays debug process output and processes keyboard input to the debug process, if any

Continue the Page Load -- When done, select Debug / Continue from the Debug menu or toolbar.

Try Pause -- From Wing, you can pause the Zope process by pressing the Pause icon in the toolbar or using Pause from the Debug menu. This is a good way to interrupt a lengthy computation to see what's going on. When done between page loads, it pauses Zope in its network service code.

Other Features -- Notice that Wing IDE's editor contains a source index and presents you with an auto-completer when you're editing source code. The Source Assistant will display context appropriate call tips and documentation. Control-click on a source symbol to jump to its point of definition (or use Goto Selected Symbol in the Source menu). Bring up the `Source Browser` from the Tools menu to look at the module and class structure of your code.

Setting Up Auto-Refresh

When you edit and save Zope External Methods or Scripts, your changes will automatically be loaded into Zope with each new browser page load.

By default, Zope Products are not automatically reloaded, but it is possible to configure them to do so. This can make debugging much faster and easier.

Take the following steps to take advantage of this feature:

- Place a file called `refresh.txt` in your Product's source directory (for example, `Products/MyProductName` inside your Zope installation). This file tells Zope to allow refresh for this product.
- Open the Zope Management Interface.
- Expand the Control Panel and Products tabs on the upper left.

- Click on your product.
- Select the Refresh tab.
- Check the „Auto refresh mode“ check box and press „Change“.
- Make an edit to your product source, and you should see the changes you made take effect in the next browser page load.

Limitations: Zope may not refresh code if you use `import` statements within functions or methods. Also, code that manages to retain references to old code objects after a refresh (for example, by holding the references in a C/C++ extension module) will not perform as expected.

If you do run into a case where auto-reload causes problems, you will need to restart Zope from the Zope Managemetn Interface's Control Panel or from the command line. Note that pressing the Stop button in Wing only disconnects from the debug process and does not terminate Zope.

Setting up Remote Debugging

Configuring Wing for remote debugging can be complicated, so we recommend using X Windows (Linux/Unix) or Remote Desktop (Windows) to run Wing IDE on the same machine as Zope but display it remotely.

When this is not possible, you can set up Wing to debug Zope running on another machine, as described below:

- **Set up File Sharing** -- You will need some mechanism for sharing files between the Zope host and the Wing IDE host. Windows file sharing, Samba, NFS, and ftp or rsync mirroring are all options. For secure file sharing via SSH on Linux, try sshfs.
- **Install Wing on Server** -- You will also need to install Wing on the host where Zope is running, if it is not already there. No license is needed for this installation, unless you plan to also run the IDE there.

  If there is no binary distribution of Wing available for the operating system where Zope is running, you can instead install only the debugger libraries as outlined in **Compiling the Wing IDE Debugger from Source**.
- **Basic Configuration** -- Follow the instructions for Single-Host Debugging above first if you have not already done so. Then return here for additional setup instructions.

- **Configure Allowed Hosts** -- You will need to add the IP address of the Zope host to the **Allowed Hosts** preference in Wing. Otherwise Wing will not accept your debug connections.

- **Configure File Mapping** -- Next, set up a mapping between the location of the Zope installation on your Zope host and the point where it is accessible on you Wing IDE host.

  For example, if your Zope host is `192.168.1.1` Zope is installed in `/home/myuser/Zope` on that machine, and `/home/myuser` is mounted on your Wing IDE host as `e:`, you would add a **Location Map** preference setting that maps `192.168.1.1` to a list containing `/home/myuser/Zope` and `file:e:/Zope`.

  For more information on this, see **File Location Maps** and **Location Map Examples** in the Wing IDE manual.

- **Modify WingDBG Configuration** -- When debugging remotely, the value given to WingDBG for the Wing Home Directory must be the location where Wing is installed on the Zope host (the default value will usually need to be changed).

- **Check Project Configuration** -- Similarly, the paths identified in Project Properties should be those on the host where Wing IDE is running, not the paths on the Zope host.

Trouble Shooting Guide

You can obtain additional verbose output from Wing IDE and the debug process as follows:

- Go into the Wing Debugging Service in the Zope Management Interface and set `Log file` under the `Configure` tab. Using `<stdout>` will cause logging information to be printed to the console from which Zope was started.

  Alternatively, set this to the full path of a log file. This file must already exist for logging to occur.

- Restart Zope and Wing and try to initiate debug.

- Inspect the contents of the log. If you are running Zope and Wing IDE on two separate hosts, you should also inspect the `error-log` file on the Wing IDE host (located in the **User Settings Directory**). It contains additional logging information from the Wing IDE process.

For additional help, send these errors logs to support at wingware.com.

Related Documents

Wing IDE provides many other options and tools. For more information:

- **Wing IDE Reference Manual**, which describes Wing IDE in detail.
- Zope home page, which contains much additional information for Zope programmers.
- **Quick Start Guide** and **Tutorial** which contain additional basic information about getting started with Wing IDE.

Verwendung von Wing IDE mit Plone

Wing IDE ist eine integrierte Entwicklungsumgebung für die Programmiersprache Python. Die Verwendung von Wing kann den Prozess des Schreibens und Debuggens von Code, der für Plone, einem leistungsfähigen Web-Content-Management-System, geschrieben ist, beschleunigen.

Da Plone auf Zope basiert, ähnelt die Einrichtung von Wing IDE mit Plone sehr der Einrichtung von Wing IDE mit Zope, welche detailliert in der **Zope Schnellstart-Anleitung** beschrieben ist. Der einzige Unterschied besteht darin, dass Sie Plone anstelle von Zope herunterladen und installieren müssen. Anstatt nur Zope zu starten, müssen Sie außerdem Plone starten (was auf win32 mit dem Plone-Startwerkzeug aus dem Startmenü gemacht wird).

Hinweise zur Leistungsfähigkeit

Plone und Zope zusammen beinhalten eine sehr große Python-Codebasis. Wenn Sie aus dem Projektmenü den Punkt `Verzeichnisbaum hinzufügen` wählen, um die gesamte Plone-Installation einzuschließen, werden Sie bedeutende CPU-intensive Verarbeitung sehen. Dies kann auf langsameren Maschinen ein Thema sein und die Durchführung kann einige Minuten in Anspruch nehmen. Dies geschieht während Wing IDE den Python Source-Code analysiert, um die Datenstrukturen zu erstellen, die für den Source-Browser, Auto-Vervollständiger, das Source-Index-Menü, die Gehe-zu-Definition und andere Funktionen des IDE's benötigt werden. Wing sollte während dieser Zeit ansprechbar bleiben, kann allerdings träge erscheinen.

So lange wie die anfängliche Analyse in Bearbeitung ist, wird der Source-Browser eine Teilmenge aller verfügbaren Code-Konstrukte enthalten und einige Funktionen, wie die Gehe-zu-Definition, werden möglicherweise solange nicht funktionieren, bis der gesamte Source-Code verarbeitet wurde.

Die meiste Arbeit wird durchgeführt, nachdem Sie zum erstem Mal die Python-Dateien

zu Ihrem Projekt hinzufügen. Danach wird ein Laufwerkspeicher verwendet, um den Zugang zu Analyseinformationen zu beschleunigen. Sie werden aber jedesmal, wenn Sie eine neue Wing IDE Sitzung starten, erneute Verarbeitung des Laufwerkspeichers sehen. Dies kann auf einigen Maschinen auch bedeutend sein.

In allen Fällen wird die Verarbeitung nach einem bestimmten Zeitraum aufhören und der Rest Ihrer Wing IDE Sitzung sollte nahzu ohne CPU-Verbrauch und selbst auf langsameren Maschinen mit einem gut aussehenden und ansprechbaren GUI laufen.

Verwandte Dokumente

Wing IDE stellt viele andere Optionen und Tools bereit. Weitere Informationen finden Sie hier:

- **Verwendung von Wing IDE mit Zope**, das beschreibt, wie Zope für die Verwendung mit Wing IDE eingerichtet wird.
- **Wing IDE Benutzerhandbuch**, das Wing IDE detailliert beschreibt.
- Plone Homepage, die Links zur Dokumentation bereitstellt.
- **Wing IDE Schnellstart-Anleitung**, die zusätzliche grundlegende Informationen darüber enthält, wie Sie Wing IDE zum Laufen bringen.

Verwendung von Wing IDE mit mod_python

Wing IDE ist eine integrierte Entwicklungsumgebung für die Programmiersprache Python. Wing kann verwendet werden, um Code zu debuggen, der durch das mod_python-Modul für den Apache-Web-Server ausgeführt wird. Dieses Dokument setzt voraus, dass mod_python installiert ist und Apache zur Nutzung konfiguriert ist. Bitte lesen Sie das Kapitel Installation des mod_python-Benutzerhandbuches, um Informationen darüber zu erhalten, wie Sie es installieren.

Da der Support für Wing's Debugger zur Zeit Single-Threaded ist, kann jeweils nur eine http-Anfrage gedebuggt werden. Für jede Anfrage wird eine neue Debug-Sitzung erstellt und die Sitzung wird beendet, wenn die Verarbeitung der Anfrage abgeschlossen ist. Wenn eine zweite Anfrage gestellt wird, während eine Anfrage gedebuggt wird, wird entweder blockiert, bis die erste Anfrage abgeschlossen ist oder sie wird ohne den Debugger verarbeitet. Dies trifft für Anfragen zu, die von einem einzelnen Python-Modul verarbeitet werden und es trifft für Anfragen zu, die von mehrfachen Python-Modulen in dem gleichen Apache-Prozess und seinen Kindprozessen verarbeitet werden. Es wird empfohlen, dass nur eine Person mod_pyhton basierende Module pro Apache-Instanz debuggt.

Schnellstart

- Kopieren Sie `wingdbstub.py` von dem Wing IDE Installationsverzeichnis entweder in das Verzeichnis, in dem das Modul ist, oder in ein anderes Verzeichnis im Python-Pfad, das vom Modul verwendet wird.
- Bearbeiten Sie `wingdbstub.py`, wenn notwendig, so dass die Einstellungen mit den Einrichtungen in Ihren Grundeinstellungen übereinstimmen. Normalerweise muss nichts eingestellt werden, es sei denn, Wing's Debug-Einstellungen wurden verändert. Wenn Sie diese Einstellungen verändern möchten, lesen Sie den Abschnitt **Remote-Debuggen** des Wing IDE Benutzerhandbuches für zusätzliche Informationen.

- Kopieren Sie `.wingdebugpw` von Ihrem **Verzeichnis der Benutzereinstellungen** in das Verzeichnis, welches das Modul, dass Sie debuggen möchten, beinhaltet. Dieser Schritt kann übersprungen werden, wenn das zu debuggende Modul auf der gleichen Maschine und unter dem gleichen Nutzer wie Wing IDE ausgeführt wird. Die `.wingdebugpw` Datei muss genau eine Zeile enthalten.

- Fügen Sie `import wingdbstub` oben im Modul, das vom mod_python-Core importiert wurde, ein.
- Fügen Sie `if wingdbstub.debugger != None: wingdbstub.debugger.StartDebug()` oben in jeder Funktion, die vom mod_python-Core aufgerufen wird, ein.
- Aktivieren Sie passives Hören in Wing IDE, indem Sie die Einstellung **Passives Hören aktivieren** auf wahr setzen.

- Starten Sie Apache neu und laden Sie eine URL, um die Ausführung des Moduls auszulösen.

Beispiel

Um das Beispiel `hello.py` aus dem Kapitel Publisher der mod_python-Anleitung zu debuggen, ändern Sie die `hello.py` Datei, so dass sie den folgenden Code enthält:

```
import wingdbstub

def say(req, what="NOTHING"):
  if wingdbstub.debugger != None:
    wingdbstub.debugger.StartDebug()
  return "I am saying %s" % what
```

Und richten Sie die mod_python-Konfigurationsanweisungen für das Verzeichnis, in dem `hello.py` ist, folgendermaßen ein:

```
AddHandler python-program .py
PythonHandler mod_python.publisher
```

Setzen Sie dann einen Haltepunkt an der Zeile `return "I am saying %s" % what`, versichern Sie sich, dass Wing auf eine Debug-Verbindung hört und laden Sie `http://[Server]/[Pfad]/hello.py` in einem Web-Browser (ersetzen Sie passende Werte für [Server] und [Pfad]). Wing sollte dann an dem Haltepunkt stoppen.

Verwandte Dokumente

- **Wing IDE Benutzerhandbuch**, das Wing IDE detailliert beschreibt.
- mod_python-Benutzerhandbuch, welches beschreibt, wie Sie mod_python installieren, konfigurieren und nutzen.

Web-CGIs mit Wing IDE debuggen

Wing IDE kann verwendet werden, um CGI-Skripte, die in Python geschrieben sind, zu debuggen, selbst wenn sie als Ergebnis des Ladens einer Seite von einem Web-Browser laufen. Um Ihre CGI's für das Debuggen mit Wing IDE einzurichten, lesen Sie den Abschnitt **Extern gestarteten Code debuggen** des Benutzerhandbuches. Achten Sie besonders auf die Berechtigungen für Dateien, insbesondere wenn Ihr Web-Server unter einem anderen Nutzer ausführt, als der Prozess, mit dem Wing IDE läuft. Sie müssen auch sicherstellen, dass die Datei `.wingdebugpw` richtig ausgewiesen ist, wie in den Anweisungen beschrieben.

Der Rest dieser Anleitung stellt einige spezielle Hinweise für das Debuggen von CGI's bereit:

1) Jeglicher Inhalt von Ihrem CGI-Skript, der vom Web-Server nicht verstanden wird, wird zum Fehlerprotokoll des Servers gesendet. Da es sehr lästig sein kann, durch alles zu suchen, ist es viel einfacher sicherzustellen, dass die gesamte Ausgabe, einschließlich der Ausgabe, die bei einem Fehler gemacht wurde, in Ihrem Web-Browser angezeigt wird.

   Um dies zu tun, fügen Sie Folgendes am Anfang Ihres Codes ein, bevor Sie `wingdbstub` importieren oder den Debugger-API aufrufen:

   ```
   print "Content-type: text/html\n\n\n<html>\n"
   ```

   Dies bewirkt, dass alle nachfolgenden Daten im Browser-Fenster einbezogen sind, selbst wenn Ihr normaler Content-Typ-Spezifikationscode nicht erreicht wird.

2) Platzieren Sie einen Catch-All Exception-Handler auf der höchsten Ebene Ihres CGI-Codes und drucken die Exception-Informationen im Browser. Die folgende Funktion ist hilfreich, um den Status der CGI-Umgebung zu prüfen, wenn eine Exception auftritt:

   ```
   import sys
   import cgi
   import traceback
   ```

```
import string

#--------------------------------------------------
-----------------
def DisplayError():
  """ Output an error page with traceback, etc """

  print "<H2>An Internal Error Occurred!</H2>"
  print "<I>Runtime Failure Details:</I><P>"

  t, val, tb = sys.exc_info()
  print "<P>Exception = ", t, "<br>"
  print "Value = ", val, "\n", "<p>"

  print "<I>Traceback:</I><P>"
  tbf = traceback.format_tb(tb)
  print "<pre>"
  for item in tbf:
    outstr = string.replace(item, '<', '&lt;')
    outstr = string.replace(outstr, '>', '&gt;')
    print string.replace(outstr, '\n', '\n'), "<BR>"
  print "</pre>"
  print "<P>"

  cgi.print_environ()
  print "<BR><BR>"
```

3) Wenn Sie `wingdbstub.py` verwenden, können Sie `kSilent=0` setzen, um zusätzliche Informationen vom Debug-Server zu erhalten, um Probleme, die zu Wing IDE zurückverbinden, zu debuggen. Diese Informationen werden in `stderr` gespeichert und sind daher in der Fehlerprotokolldatei des Web-Servers zu finden.

4) Wenn Sie den gesamten Debugger-API nutzen, können Sie Ihr `CErrStream` Objekt setzen, um Ausgabe entweder an `stdout`, `stderr` oder jeden anderen Dateistrom zu senden. Verwenden Sie dies, um Fehler an den Browser, das Fehlerprotokoll des Web-Servers beziehungsweise an eine Datei zu senden.

5) Wenn Sie nicht in der Lage sind, Skript-Ausgabe, die für die Fehlerbehebung relevant sein kann, zu sehen, versuchen Sie, Ihr CGI-Skript von der Befehlszeile aufzurufen. Das Skript kann fehlschlagen, aber Sie können Nachrichten vom Debug-Server sehen, wenn diese aktiviert sind.

6) Wenn alles andere scheitert, lesen Sie die Dokumentation Ihres Web-Browsers, um dessen Fehlerprotokoll zu ermitteln und zu lesen. In Linux

mit Apache ist dies oft in `/var/log/httpd/error_log`. Alle Fehler, die nicht im Browser sichtbar sind, werden dort angehängt.

7) Wenn der Debugger einmal für ein CGI-Skript funktioniert, müssen Sie den `wingdbstub` Import in jedem und jedem zweiten Top-Level CGI in der gleichen Weise einrichten. Da dies ein wenig langwierig sein kann und da der Import am Anfang von jeder Datei geschehen muss (in dem `__main__` Bereich), ist es sinnvoll, Ihren Code so zu entwickeln, dass alles Laden von Seiten für eine Website durch ein Single-Entry-Punkt-CGI geschieht und seitenspezifisches Verhalten wird mittels der Abfertigung innerhalb des CGI zu anderen Modulen erhalten. Mit Python's flexiblen Import- und Anforderungsfunktionen ist dies relativ einfach zu erledigen.

Wing IDE for OS X

- Wing IDE uses X windows on OS X, but support for X is not by default part of all versions of OS X. Thus you also need to obtain and install an X server and X Window manager. There are a number of options for this:

  (1) Apple's X11 Server for OS X is among the fastest and best integrated options. It includes both the X Server and a native Aqua window manager, although you can replace the default window manager with your favorite if you wish. Apple X11 Server comes with OS X 10.3 and later, but is not installed by default and must be installed separately from the installation CDs. X11 is part of optional installs package on the installation disk; often it's hidden by default so you'll need to scroll down in the finder window for the installation disk to find it. For OS X 10.3, X11 is also available as downloadable package from Apple's website, but this version will not work with OS X 10.4.

  (2) XDarwin (1.1 or later) can be used together with the window manager of your choice. Window Maker is one that users have reported as working well. OroborOSX (0.75a4r2 or later) also works but can be quite slow in comparison with other options (as of 0.8 preview 2). Note that for some versions of OroborOSX, you need to unpack both the top-level OroborOSX tar file and the XDarwin.tar file located inside the installation.

- Now you're ready to install Wing IDE. Just download Wing IDE, unpack it and move it into place.

  Double clicking on the Wing IDE `tar` archive will expand its contents into a new folder on disk in the same location as the archive. Subsequently, the `tar` archive can be removed and the expanded form of the application can be moved on disk as desired.

  Note: One some systems, tar will truncate file names leading to a broken installation. In that case, unpack the archive with `gnutar` instead.

- Next make sure your X Windows server chosen above is running and set up to allow connections from X clients on `:0.0`. Wing will automatically start Apple X11 Server if it is present and not yet running.

- Then double click on the Wing IDE app folder created by unpacking the `tar` archive. The first time you start Wing, it will ask you to accept the license agreement and will ask for a license. Use the first (default) option in the dialog that appears to obtain a 10 day trial license (this can be renewed twice when the trial period expires).

  At this time, Wing will create a **User Settings Directory** in `~/.wingide2`, which is used to store preferences and other settings.

- The location of Wing's internal application contents folder is referred to as `WINGHOME`. For example if you unpacked Wing into `/Applications/Wing` then `WINGHOME` will be `/Applications/Wing/WingIDE.app/Contents/MacOS`.

  To start Wing from the command line, execute `wing-personal` located in `WINGHOME` (`/Applications/Wing/WingIDE.app/Contents/MacOS/wing` in the above example). When you do this, you may need to set your `DISPLAY` environment variable to point to your X Server (for example `setenv DISPLAY :0.0`).

- See the **Tutorial** and **Wing IDE Quickstart Guide** for additional information on getting starting with Wing IDE.

Usage Tips

- Wing starts with a keyboard mapping that emulates the most commonly used key standards on the Macintosh. This mapping will not work properly unless you uncheck the „Enable Key Equivalents under X11“ preference in Apple X11 Server configuration. Wing will warn when this option is checked, but may fail to do so under other X11 server software. Wing also tries to detect if an Apple keyboard is in use, but you may need to set the **Apple Keyboard** preference to `yes` or `no` if the detection fails.

  You can alter keyboard mapping (for example, to use Emacs bindings instead) with the **Personality** preference, or change individual key mappings with the **Custom Key Bindings** preference. If you do use a keyboard personality other than the OS X personality, you may want to map the option or command key to the Alt modifier using the **Global X11 Alt Key** preference. Note that this preference affects all X11 applications, not just Wing.

- Under OS X 10.4 (Tiger), the option/compose key used to enter accented and other foreign characters will not work because of changes in the Apple X11 applications. To fix this, enable the **Fix Option key bug in Tiger (OS X 10.4)** preference. This will affect all X11 applications, not just Wing.

If you are running other X11 applications and want to workaround this bug yourself, you'll want to either enable the XKEYBOARD extension on the X11 server or use xmodmap to assign the Mode_Switch key to a modifier other than mod1. The xmodmap script that Wing runs when the **Fix Option key bug in Tiger (OS X 10.4)** preference is enabled is ${WINGHOME}/resources/osx/fix-tiger-mode-switch.xmodmaprc. The script removes Mode_Switch from mod1 and adds it to mod5 and is run only if the preference is enabled and xmodmap reports that Mode_Switch is assigned to mod1.

If the X11 Application „Use the system keyboard layout" preference is enabled, then the X11 server may modify its keyboard mapping when the system keyboard changes. You may need to disable this preference or restart Wing after the keyboard layout changes because Wing will not re-apply the fix after the X11 keyboard changes. This should only be an issue if you change keyboard layouts while Wing is running.

- Although Wing is not a native OS X application, it starts up with a display theme that tries to match the OS X look and feel (font size often needs to be altered from the **Display Font/Size** preference). Additional display themes can be selected from the **Display Theme** preference.

  It is also possible to download other themes for GTK2 and place them into `Contents/MacOS/bin/gtk-bin/share/themes` inside your Wing IDE installation. Once this is done and Wing is restarted, they will show up in the **Display Theme** preference.

  One nice OS X like theme is AquaX, currently not included with Wing because we cannot redistribute it under its licensing.

  Note that only themes that do not use a theme engine or use one of Redmond, Smooth, or Pixmap will work with Wing IDE. We cannot make any guarantees for performance or results when using themes not included with Wing IDE, although Aqua X is known to work well.

- When using the **Zope Support Module** or following instructions that refer to `WINGHOME` note that `WINGHOME` is defined as the location of the `wing` executable, which on Mac OS X is inside the `Contents/MacOS` folder of the Wing IDE app folder. E.g., if you unpacked Wing into `/Applications/Wing` then `WINGHOME` will be `/Applications/Wing/WingIDE.app/Contents/MacOS`.

- Right-click for menus by holding down the Option/Alt key while clicking. Middle-click by holding down the Control key while clicking. These defaults can be changed in your X11 server's preferences. For example, under Apple X11 Server, change so Option/Alt is button two and Control is button three with this command:

```
defaults write com.apple.x11 fake_button2 option
defaults write com.apple.x11 fake_button3 control
```

Or change so that Option/Alt is button two and Apple/Command is button three:

```
defaults write com.apple.x11 fake_button2 option
defaults write com.apple.x11 fake_button3 command
```

Then restart the X11 Server.

- You can configure Apple X11 Server to automatically transfer focus to the window the mouse pointer is over, or to pass through the click that is used to bring focus to the window so it is also processed by the application.

  To move focus with the mouse pointer:

  ```
  defaults write com.apple.x11 wm_ffm true
  ```

  To pass through the focus click:

  ```
  defaults write com.apple.x11 wm_click_through -bool true
  ```

  You must restart Apple X11 changing either of these configurations before they take effect.

  Other configuration options like this can be obtained by looking in the manual pages for quartz-wm and Xquartz:

  ```
  man quartz-wm
  man Xquartz
  ```

Known Problems

There are some known problems resulting from platform-specific behaviors on OS X:

- There is no way to control-click, so control-left-click for the goto-definition feature doesn't work. Instead, use Goto Selected Symbol from the Source menu instead; this works relative to position of the insertion cursor in the current editor and can be accessed by the key binding shown in the Source menu.

Please send bug reports to bugs at wingware.com.

Related Documents

- **Wing IDE Quickstart Guide**, which contains additional information about getting started with Wing IDE.
- **Other How-Tos**, for getting started with Wing IDE and specific tools.
- **Wing IDE Reference Manual**, which describes Wing IDE in detail.

Using Wing IDE with pygame

Wing IDE is an integrated development environment for the Python programming language. Wing can be used to speed up the process of writing and debugging code that is written for pygame, an open source framework for game development with Python..

To get started, refer to the tutorial in the `Help` menu in Wing and/or the **Wing IDE Quickstart Guide**.

Debugging pygame

You should be able to debug pygame code with Wing just by starting debug from the IDE. However, pygame running in full screen mode will not work properly and may crash Wing. Use window mode instead.

This problem exists with other Python IDEs as well; we have not yet determined what the cause is.

Related Documents

Wing IDE provides many other options and tools. For more information:

- **Wing IDE Reference Manual**, which describes Wing IDE in detail.
- **Wing IDE Quickstart Guide** which contains additional basic information about getting started with Wing IDE.

Große Werte und Strings im Debugger verarbeiten

Um das Aufhängen an großen Werten während dem Schreiten durch den Code und anderen Debugger-Aktionen zu vermeiden, begrenzt der Debugger die Größe der Konstrukte, die er anzeigen wird.

Sie können die Größenbegrenzungen für große, zusammengesetzte Werte mit der Einstellung **Große Listenschwelle** ändern. Wenn Sie das machen, müssen Sie wahrscheinlich auch die Geduld des Debuggers für das Warten auf diese großen Listen zur Übertragung mit der Einstellung **Netzwerkabschaltung** erhöhen. Dieser Wert ist jetzt absichtlich niedrig eingestellt, um überlanges Aufhängen des IDE's zu verhindern, wenn das Debug-Programm abstürzt.

Lange Strings werden standardmäßig auch abgeschnitten, wenn Sie vom Debug-Prozess zum IDE geschickt werden. Um einen abgeschnittenen String zu erweitern, klicken Sie auf ihn und sehen sich seine vollständige Form im Textansichtsbereich am Ende des Debugger-Fensters an. Wenn Sie in einer verkleinerten Ansicht arbeiten, klicken Sie stattdessen mit der rechten Maustaste auf den String und verwenden den Eintrag `Anzeige in Textansicht` von dem Popup-Menü, das erscheint.

Die maximal anzeigbare Länge von Strings wird mit der Einstellung **Große String-Schwelle** kontrolliert.

Eine Möglichkeit, große Werte anzusehen, ohne die Einstellungsschwellen zu erhöhen, besteht darin, einen Ausdruck in die Werkzeuge Beobachten oder Debug-Test einzugeben, welche aus dem Menü Werkzeuge verfügbar sind. Sie können zum Beispiel für einen großen Bereich einen Wert wie `a[2:5][7]` eingeben, um zu einer kontrollierbaren Wertgröße zu kommen.

Trotz der pro-Wert Größenbegrenzungen ist es immer noch möglich, große Mengen von Daten vom Debug-Server zu übertragen. Wir empfehlen allerdings etwas Vorsicht anzuwenden, wenn Sie die Option `Mehr erweitern` in der Textansicht nutzen.

C/C++ und Python zusammen debuggen

Der Wing IDE Debugger funktioniert zur Zeit nur für Python-Code und kann es selbst nicht handhaben, in C oder C++ zu schreiten. Sie können jedoch den VC++ oder den gdb-Debugger in Verbindung mit dem Wing IDE Debugger einrichten, um Fehler entweder in C oder Python zur gleichen Zeit zu debuggen.

Dies wird gemacht, indem Ihr Python-Code unter dem VC++ oder gdb-Debugger ausgeführt wird, wie Sie es für C/C++ Debuggen sowieso machen würden, während Sie den Wing Debugger zur gleichen Zeit verwenden, indem Sie `wingdbstub` in Ihren Code importieren.

Um den C/C++ Code zu debuggen, müssen Sie mit einer Kopie von Python ausführen, die vom Source-Code mit Debug-Symbolen kompiliert wurde. Zur Konfiguration von `wingdbstub` lesen Sie bitte den Abschnitt **Extern gestarteten Code debuggen** im Benutzerhandbuch.

Werfen Sie auch einen Blick auf die zusätzlichen Informationen zu **gdb und Wing zusammen verwenden**. Die Verwendung von Wing und VC++ ist weniger anfällig für Probleme und hat daher zur Zeit kein eigenes How-To.

Erweiterungsmodule in Linux/Unix debuggen

Gdb kann als ein Tool verwendet werden, dass beim Debuggen von C/C++ Erweiterungsmodulen, die für Python geschrieben sind, hilft, selbst wenn die Verwendung ein wenig kompliziert und anfällig für Probleme sein kann. Der folgende Abschnitt enthält einige Tipps, um die Anwendung zu erleichtern.

Dieser Abschnitt setzt allerdings voraus, dass Sie mit gdb bereits vertraut sind. Weitere Informationen über gdb-Befehle finden Sie in der gdb-Dokumentation.

Der erste Schritt beim Debuggen von C/C++ Modulen mit gdb ist, sich zu versichern, dass Sie eine Python-Version verwenden, die mit Debug-Symbolen kompiliert wurde. Um dies festzustellen, benötigen Sie eine Source-Distribution von Python und Sie müssen die Distribution, wie in der beiliegenden `README`-Datei beschrieben, konfigurieren.

In den meisten Fällen kann dies folgendermaßen gemacht werden: (1) Geben Sie `./configure` ein, (2) geben Sie `make OPT=-g` ein oder bearbeiten Sie das Makefile, so dass `OPT=-g`, (3) geben Sie `make` ein und (4) wenn der Build abgeschlossen ist, installieren Sie ihn mit `make install` (aber lesen Sie zuerst die `README`-Datei, wenn Sie nicht in `/usr/local/lib/python` installieren möchten).

Wenn Sie ein Erweiterungsmodul erstellen, dass Sie in den Python-Interpreter kompilieren, können Sie jetzt Python einfach innerhalb von gdb ausführen, einen Haltepunkt an der gewünschten Stelle in Ihrem Erweiterungsmodul setzen und Ihr Python-Testprogramm ausführen.

In den meisten Fällen wird das Erweiterungsmodul jedoch nicht in Python kompiliert, sondern wird stattdessen dynamisch in der Laufzeit geladen. Um Ihr Erweiterungsmodul zu laden, muss es sich im `PYTHONPATH` oder innerhalb des gleichen Verzeichnisses, in dem das Modul in die Python-Source `import`-iert wurde, befinden.

Gdb verlangt zusätzlich, dass `LD_LIBRARY_PATH` so eingestellt wird, dass es das Verzeichnis beinhaltet, in dem das dynamisch geladene Modul platziert ist. Ein allgemeines Problem dabei ist, dass gdb jedesmal wenn es läuft, `.cshrc` neu lesen wird, so dass die

Einstellung von `LD_LIBRARY_PATH` vor dem Aufrufen von gdb keinen Effekt hat, wenn Sie auch `LD_LIBRARY_PATH` in `.cshrc` setzen. Um dies zu umgehen, setzen Sie stattdessen `LD_LIBRARY_PATH` in `.profile`. Diese Datei wird nur einmal bei der Anmeldung gelesen.

Als nächstes sollten Sie Ihre `~/.gdbinit` Datei einrichten, indem Sie die Inhalte der Datei `Misc/gdbinit` von der Python Source-Distribution kopieren. Dies enthält einige nützliche Makros, um Python-Code von gdb zu überprüfen. Sie sollten auch das folgende hinzufügen, welches Sie den Python-Stack drucken lässt:

```
define ppystack
  while $pc < Py_Main || $pc > Py_GetArgcArgv
    if $pc > eval_frame && $pc < PyEval_EvalCodeEx
      set $__fn = PyString_AsString(co->co_filename)
      set $__n = PyString_AsString(co->co_name)
      printf "%s (%d): %s\n",  $__fn, f->f_lineno, $__n
    end
    up-silently 1
  end
  select-frame 0
end
```

Starten Sie dann Python wie folgt:

```
myhost> gdb
(gdb) file python
(gdb) run yourprogram.py yourargs
```

Beachten Sie, dass Haltepunkte in einer gemeinsamen Bibliothek nicht gesetzt werden können, bis die gemeinsame Bibliothek geladen ist. Wenn die Ausführung Ihres Programms das Laden Ihrer Erweiterungsmodul-Bibliothek auslöst, können Sie `^C^C` verwenden, um das Debug-Programm zu unterbrechen, Haltepunkte zu setzen und dann fortzufahren.

Andernfalls müssen Sie die Ausführung Ihres Programms fortsetzen, bis das Erweiterungsmodul geladen ist. Im Zweifelsfall fügen Sie eine `print` Anweisung am Punkt des Imports hinzu oder Sie können eine Haltepunkt an `PyImport_AddModule` setzen (dies kann nach `file python` und vor der Ausführung gesetzt werden, da dieser Aufruf nicht in einer gemeinsamen Bibliothek ist).

Selbst wenn Sie alle oben genannten Schritte ausführen, werden einige Versionen von gdb leider oft durcheinander kommen, wenn Sie gemeinsame Bibliotheken während einer einzelnen Debug-Sitzung wiederholt laden und entladen. Sie können normalerweise

Python 5-10 Mal wiederholen, aber danach wird es wahrscheinlich abstürzen oder Sie werden Fehler beim Stoppen an Haltepunkten oder anderes merkwürdiges Verhalten bemerken. Wenn dies auftritt, gibt es keine andere Alternative, als gdb zu beenden und neu zu starten.

Abschließend ein Tipp für die Anzeige von Python-Daten von der C/C++ Seite, wenn Sie gdb verwenden. Der folgende gdb-Befehl wird die Inhalte von einem `PyObject *` mit dem Namen `obj` ausdrucken, so als wenn Sie den Befehl `print obj` innerhalb der Programmiersprache Python ausgeführt hätten:

```
(gdb) p PyObject_Print (obj, stderr, 0)
```

Anwendungen debuggen, die XGrab* benutzen

Wing versucht nicht, den XGrabPointer oder XGrabKey und ähnliche Ressourcen-Greifer zu unterbrechen, wenn Ihr Debug-Prozess anhält. Das bedeutet, dass X in einigen Debug-Fällen möglicherweise auf die Tastatur, die Maus oder beides nicht reagiert.

Den Debugger zu fixieren, so dass er Ressourcen erkennt und loslässt, ist in diesem Fall ziemlich schwierig und anfällig für verwirrendes und unerwünschtes Verhalten. Hier sind einige Hinweise, wie Sie dieses Problem umgehen können:

- Einige Toolkits haben eine Option, um Ressourcen-Greifer zu deaktivieren, insbesondere um dieses Problem während dem Debuggen zu vermeiden. PyQt hat zum Beispiel die Option `-nograb` in der Befehlszeile, die verhindert, dass jemals die Tastur oder der Mauszeiger festgehalten werden. Das Hinzufügen dieser Option zur Befehlszeile des Debug-Prozesses löst das Problem.

  Wenn Sie Ihre eigenen Aufrufe an `XGrab*` oder ähnliche Funktionen schreiben, bedenken Sie, einen Modus hinzuzufügen, damit diese Aufrufe übersprungen werden.

  Ein Trick, der oft hilft, ist, die Verarbeitung zu verschieben, und zwar vom Callback, bei dem der Mauszeiger- oder Tastatur-Greifer noch wirksam ist, in ein asynchrones Callback, dass während der Leerlaufzeit auftritt. Verwenden Sie zum Beispiel unter GTK `gtk_idle_add()` und versuchen Sie in wxPython einen wxTimer.

- Unter XFree 4.2 oder höher gibt es eine Konfigurationsoption, die Sie einstellen können, so dass Strg-Alt-Keypad-/ durch jeden aktiven Mauszeiger- und Tastatur-Greifer brechen wird:

  ```
  # Let user break server grabs with strg-alt-keypad-/
  Option "AllowDeactivateGrabs" "true"
  ```

  Dies geht in Ihre XF86Config-Datei im Abschnitt „ServerOptions“. Prüfen Sie `man XF86Config` auf den Suchpfad, den X verwendet, um die Konfigurationsdatei zu finden und diejenige zu finden, die auf Ihrem System verwendet wird. Auf Mandrake 8.2 zum Beispiel ist die Konfigurationsdatei `/etc/X11/XF86Config-4`.

Sie müssen Ihren X Server neu starten, damit die Konfigurationsänderungen wirksam werden (zum Beispiel: Melden Sie sich ab und wieder an). Seien Sie sich bewußt, dass Änderungen in Ihrer XF86Config-Datei Fehler beim Start von X verursachen könnnen. Wenn dies passiert, müssen Sie es im Textmodus fixieren. Wenn Sie in diese Situation kommen, dann ist die Eingabe von `startx` nach jeder Bearbeitung ein guter Weg, um zu überprüfen, ob Ihre Fehlerbehebung funktioniert.

Wenn Sie überprüfen müssen, welche Version von XFree Sie installiert haben, dann funktioniert normalerweise die Eingabe von `rpm -q XFree86` oder Sie verwenden `man XFree86`, was die Versionsnummer am Ende der Handbuchseite anzeigt.

- Wenn in Linux alles andere scheitert, können Sie in den meisten Distributionen `Strg-Alt-F1` bis `Strg-Alt-F6` verwenden, um sechs virtuelle, nur-Text Konsolen zu erhalten. Von dort aus können Sie `ps` ausführen, um den Debug-Prozess zu finden und mit `kill -TERM` oder `kill -9` zu löschen, wenn erforderlich. Dies wird Ihre X Windows-Anzeige entsperren, zu der Sie mit `Strg-Alt-F7` zurückkehren können.

- Die Anzeige Ihres Debug-Prozesses auf einem anderen Bildschirm vermeidet auf diese Weise, dass Wing gebunden wird. Die meisten Server entsperren den Bildschirm, sobald Sie den Debug-Prozess von Wing löschen.

Nicht-Python-Hauptschleifen debuggen

Aufgrund der Art und Weise wie der Python-Interpreter das Debuggen unterstützt, kann der Debug-Prozess unansprechbar werden, wenn Ihr Debug-Prozess für lange Perioden in nicht-Python-Code, wie C oder C++, läuft. Immer wenn der Python-Interpreter über lange Zeiträume nicht aufgerufen wird, kann es sein, dass Nachrichten von Wing IDE völlig ignoriert werden und das IDE kann die Verbindung zum Debug-Prozess trennen, so als ob es abgestürzt ist. Dies beeinflusst in erster Linie das Anhalten eines laufenden Debug-Programms oder das Setzen, Löschen oder Bearbeiten von Haltepunkten während dem Ausführen eines Debug-Programms.

Beispiele für Umgebungen, die signifikante Zeiträume außerhalb des Python-Interpreters verbringen, schließen GUI-Kits, wie Gtk, Qt, Tkinter, wxPython und einige Web-Entwicklungswerkzeuge wie Zope ein. Zur Vereinfachung für diesen Abschnitt, nennen wir sie hier „nicht-Python-Hauptschleifen“.

Unterstützte Nicht-Python-Hauptschleifen

Wing unterstützt bereits Gtk, Tkinter, wxPython und Zope. Wenn Sie eins von diesen verwenden, nutzen Sie gar keine nicht-Python-Hauptschleife. Sie brauchen diesen Abschnitt dann nicht weiterlesen.

Mit Nicht-Python-Hauptschleifen arbeiten

Wenn Sie eine nicht unterstützte nicht-Python-Hauptschleife verwenden, die normalerweise Python-Code für längere Zeitperioden nicht aufruft, können Sie dieses Problem umgehen, indem Sie Code zu Ihrer Anwendung hinzufügen, der dafür sorgt, dass Python-Code regelmäßig aufgerufen wird. Unter PyQt fügen Sie zum Beispiel den folgenden

Code hinzu, nachdem Sie Ihre `QApplication` erstellt haben und bevor Sie `exec_loop()` aufrufen:

```
# Hack to burn some Python bytecode periodically so Wing's
# debugger can remain responsive while free-running
timer = QTimer()
def donothing(*args):
  for i in range(0, 100):
    x = i
timer.connect(timer, SIGNAL("timeout()", donothing)
timer.start(500, 0)
```

Ähnlicher Code kann in den meisten Umgebungen von nicht-Python-Hauptschleifen geschrieben werden.

Die Alternative zum Ändern Ihres Codes ist, einen speziellen Plug-in Support für den Wing Debugger zu schreiben, der verursacht, dass die Debug-Server-Sockets bedient werden, selbst wenn Ihr Debug-Programm in nicht-Python-Code läuft. Der Rest dieses Abschnittes beschreibt, was Sie wissen müssen, um dies umzusetzen.

Nicht-Python-Hauptschleifen Internals

Wing verwendet eine Netzwerkverbindung zwischen dem Debug-Server (dem Debug-Prozess) und dem Debug-Client (Wing IDE), um den Debug-Prozess vom IDE zu kontrollieren und um das IDE zu informieren, wenn Ereignisse (wie zum Beispiel das Erreichen eines Haltepunktes oder einer Exception) im Debug-Prozess auftreten.

Solange wie das Debug-Programm an einem Haltepunkt oder einer Exception angehalten oder gestoppt ist, bleibt die Kontrolle beim Debugger und er kann auf Anfragen vom IDE antworten. Wenn das Debug-Programm jedoch läuft, wird der Debugger selbst nur aufgerufen, solange Python-Code vom Interpreter ausgeführt wird.

Dies ist normalerweise kein Problem, weil die meisten laufenden Python-Programme viel Python-Code ausführen! In einer nicht-Python-Hauptschleife kann das Programm jedoch komplett in C, C++ oder einer anderen Sprache bleiben und den Python-Interpreter für einen langen Zeitraum überhaupt nicht aufrufen. Infolgedessen hat der Debugger keine Möglichkeit, Anfragen vom IDE zu bedienen. Anhalten oder das Anhängen von Anfragen und neuen Haltepunkten können in diesem Fall komplett ignoriert werden und das IDE kann vom Debug-Prozess abtrennen, da dieser nicht reagiert.

Wing handhabt dies, indem es seine Netzwerk-Sockets in jede der unterstützten nicht-Python-Hauptschleifen installiert, wenn sie als anwesend im Debug-Programm erkannt

werden. Wenn die Sockets registriert sind, wird die nicht-Python-Hauptschleife in Python-Code zurückkehren, immer wenn es unerledigte Netzwerkanfragen gibt.

Unterstützung für Nicht-Python-Hauptschleifen

Für diejenigen, die eine nicht unterstützte nicht-Python-Hauptschleife verwenden, stellt Wing ein API für das Hinzufügen der Hooks bereit, die notwendig sind, um sicherzustellen, dass die Netzwerk-Sockets des Debuggers jederzeit bedient werden.

Wenn Sie Support für eine nicht-Python-Hauptschleife schreiben möchten, müssen Sie zuerst überprüfen, ob es Hoffnung gibt, das Socket des Debuggers in dieser Umgebung zu registrieren. Jede Hauptschleife, die bereits UNIX/BSD-Sockets `select()` aufruft und die für erweiterbare Socket-Registrierung entworfen ist, wird funktionieren und ist einfach zu unterstützen. Gtk und Zope fallen beide in diese Kategorie.

In anderen Fällen kann es erforderlich sein, Ihren eigenen `select()` Aufruf zu schreiben und die Hauptschleife zu überlisten, dieses periodisch aufzurufen. Dies ist die Art und Weise, wie die Tkinter- und wxPython-Hooks funktionieren. Einige Umgebungen können zusätzlich erfordern, einigen nicht-Python Schnittstellen-Code zu schreiben, wenn die Umgebung nicht bereits dafür eingerichtet ist, in Python-Code zurückzukehren.

Hauptschleifen-Hooks sind als separate Module geschrieben, die in `src/debug/server` innerhalb von `WINGHOME` platziert sind. Das Modul `_extensions.py`, das auch dort zu finden ist, beinhaltet eine allgemeine Klasse, die die API-Funktionen, die für jedes Modul erforderlich sind, definiert und die der Ort ist, an dem neue Module registiert werden müssen (in der Konstanten `kSupportedMainloops`).

Support für Nicht-Python-Hauptschleifen schreiben

Um Ihren eigenen Support für nicht-Python-Hauptschleifen hinzuzufügen, müssen Sie folgende Schritte ausführen:

1) Kopieren Sie eines Ihrer Source-Beispiele (wie `_gtkhooks.py`), die in `src/debug/server` zu finden sind, als ein Framework zum Schreiben Ihrer Hooks. Nennen Sie Ihr Modul etwa so: `_xxxxhooks.py`, wobei `xxxx` der Name der Umgebung Ihrer nicht-Python-Hauptschleife ist.

2) Implementieren Sie die Methoden `_Setup()`, `RegisterSocket()` und `UnregisterSocket()`. Ändern Sie keinen Code aus den Beispielen, außer dem Code in den Methoden. Die Namen der Klassen und Konstanten in der höchsten Ebene der Datei müssen gleich bleiben.

3) Fügen Sie den Namen Ihres Moduls, ohne das `'.py'` zu der Liste `kSupportedMainloops` in `_extensions.py` hinzu.

Beispiele von bestehenden Support-Hooks für nicht-Python-Hauptschleifen können in `src/debug/server` innerhalb von `WINGHOME` gefunden werden.

Wenn Sie beim Schreiben Ihrer Hooks für die nicht-Python-Hauptschleife Schwierigkeiten haben, kontaktieren Sie bitte den Technischen Support unter http://wingware.com/support. Wir helfen Ihnen gern weiter und begrüßen jeden Beitrag von Hooks, die Sie schreiben.

Debuggen mit Python, das mit --with-pydebug kompiliert wurde

Das dbgtracer-Modul, das mit Wing IDE geliefert wird, wird nicht unter einer Kopie von Python, die mit der Option `--with-pydebug` kompiliert wurde, laufen, weil das `InitModule` Symbol absichtlich anders definiert ist, wenn Python mit dieser Option kompiliert wird. Sehen Sie am Ende von `Include/modsupport.h` in Ihrem Python Source-Code für Einzelheiten.

Die einzige Lösung ist, das dbgtracer-Erweiterungsmodul von Wing gegen Python-Sources, die mit der Option `--with-pydebug` konfiguriert sind, neu zu kompilieren. Die Neukompilierung des Debug-Servers ist im **Kapitel Debugger** des Handbuches beschrieben.

Beachten Sie, dass die Ausführung des IDE's mit einer Kopie von Python, die auf diese Weise kompiliert wurde, erfordert, dass Sie es mit der `--verbose` Option (Posix) ausführen oder `console_wing.exe` (Windows) verwenden, so dass Sie auf die `Print left references? [ny]` Anfragen reagieren können, die in dem Konsolefenster, von dem Sie Wing IDE gestartet haben, und in dem Konsolefenster für den Debug-Prozess erscheinen.

Debugging Code Running Under Py2exe

Sometimes it is useful to debug Python code launched by an application produced by `py2exe` -- for example, so that users can debug Python scripts for the app. When `py2exe` produces the `*.exe`, it strips out all but the modules it thinks will be needed by the application and may miss any required by scripts added after the fact. For Wing's debugger this is an issue because it requires certain modules to function properly.

By including these imports in the application's Python code, Wing's debugger will work:

```
import socket
import select
import traceback
import struct
import cPickle
import site
import string
```

This forces `py2exe` to bundle these needed modules with the `.exe`, and then wingdbstub can launch the netserver and establish a network connection to the IDE.

The above was tested with `py2exe` run with the `bundle=2` option.

For further information see **Debugging Externally Launched Code**